МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

отчет

по лабораторной работе №5

по дисциплине «Алгоритмы и структуры данных»

Тема: Кодирование и декодирование

Студент гр.8304 _____ Холковский К.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

Задание.

Вариант 5

Кодирование: динамическое Хаффмана

Цель работы.

Написать программу, кодирующую данные из полученного файла адоптивным алгоритмом Хаффмана.

Описание алгоритма.

После считывания содержимого применяем к каждому символу поочередно следующий алгоритм. Строится бинарное дерево, в котором изначально находится 1 пустой лист, далее проверяем на наличие нашего символа в дереве, если его нет, то выводим путь до пустого листа и сам символ, затем на месте пустого листа строим конструкцию из узла-счетчика пустого листа и нашего символа, перестраиваем дерево по правилам хранения в нем. Если же элемент уже находился в дереве, то выводим путь до него в дереве и перестраиваем дерево.

Описание функций и структур данных.

`Struct kanocha` – нужна для хранения информации о листе (символ, кол-во символов, родитель, дети).

`Struct bin_tree` - нужна для хранения массива листов в дереве

`size_t find(const char a)` - ищет элемент в дереве

`void add(const char a)` - добавляет новый элемент в дерево

`void swap(int one, int second)` - меняет местами два листа

`void rebuild(const char a)` - перестраивает дерево по правилам хранения в нем

`void code(std::ifstream& in, std::ofstream& out)` - кодирует символы из in в out, используя выше перечисленные методы.

Выводы.

В ходе выполнения данной работы была написана программа, кодирующая полученные данные методом хаффмана.

Тестирование

Пример вывода программы

easy test	**e0a00s100y000 1100t1101101001**
abbaabbaabb	**a0b0101010101010101**

Исходный код

```
#include <iostream>
#include <fstream>
#include <vector>
#include <variant>
#include <algorithm>

class bin_tree {
struct kanoha {
     kanoha(): symbol('\0'), count(0), parent(-1), left(-1), right(-1) {}
     kanoha(char b, int par): symbol(b), count(0), parent(par), left(-1),
right(-1) {}
     char symbol;
     size_t count;
     int parent;
     int left;
     int right;
};

std::vector<kanoha> arr;

size_t find(const char a) {
     for(size_t i = 0; i < arr.size(); ++i) {
           if(arr[i].symbol == a) {
                 return i;
           }
     }
     return arr.size() - 1;
}

void add(const char a) {
     arr[arr.size() - 1].left = arr.size() + 1;
     arr[arr.size() - 1].right = arr.size();
     arr.push_back(kanoha(a, arr.size() - 1));
```

```
	arr.push_back(kanoha());
	arr[arr.size() - 1].parent = arr.size() - 3;
}

void swap(int one, int second) {
	if(one == second) return;
	std::swap(arr[one].symbol, arr[second].symbol);
	std::swap(arr[one].count, arr[second].count);
	std::swap(arr[one].left, arr[second].left);
	std::swap(arr[one].right, arr[second].right);
	if(arr[one].right != -1)
		arr[arr[one].right].parent = one;
	if(arr[one].left != -1)
		arr[arr[one].left].parent = one;
	if(arr[second].right != -1)
		arr[arr[second].right].parent = second;
	if(arr[second].left != -1)
		arr[arr[second].left].parent = second;
}

void rebuild(const char a) {
	size_t num = find(a);
	if(num + 2 == arr.size() && arr[num].count == 0) {
		arr[num].count++;
		--num;
	}
	while(true) {
		arr[num].count++;
		if(num == 0) {
			return;
		}
		int i = 0;
		while(arr[num].count > arr[num - i - 1].count)
			++i;
		if((size_t)arr[num].parent + i == num) {
			num = num - i;
		}
		else {
			swap(num, num - i);
			num = arr[num - i].parent;
		}
	}
}
```

```
public:

bin_tree() {
     arr.push_back(kanoha());
}


void code(std::ifstream& in, std::ofstream& out) {
     std::string cur_text, cur_str;
     while (std::getline(in, cur_str)) {
           cur_text += cur_str;
           cur_text += '\n';
     }
     if(cur_text.size()) {
           cur_text.pop_back();
     }
     int cur_parent, num;

     for(char ch: cur_text) {
           num = find(ch);
           std::string stack;
           while(num != 0) {
                 cur_parent = arr[num].parent;
                 if(arr[cur_parent].left == num) stack += '0';
                 else stack += '1';
                 num = cur_parent;
           }
           for(auto i = stack.rbegin(); i != stack.rend(); ++i) {
                 out << *i;
           }
           if(find(ch) + 1 == arr.size()) {
                 add(ch);
                 out << ch;
           }
           //out << " ";
           rebuild(ch);
     }
}
};

int main(int argc, char* argv[])
{
bin_tree my_tree;
```

```
if(argc == 3) {
     std::ifstream in(argv[1]);
     std::ofstream out(argv[2]);
     if(!in.is_open() || !out.is_open()) {
           std::cout << "Не удалось открыть файл:(" << std::endl;
           return 0;
     }
     my_tree.code(in , out);
     std::cout << "Дело сделано!" << std::endl;
}
else {
     std::cout << "Введите имена файлов, откуда и куда кодировать!" <<
std::endl;
}
return 0;
}
```